**import** nltk

nltk**.**download("punkt")

**from** nltk.stem.lancaster **import** LancasterStemmer

stemmer **=** LancasterStemmer()

*# Importing Libraries needed for Tensorflow processing*

**import** tensorflow **as** tf *#version 1.13.2*

**import** numpy **as** np

**import** tflearn *#version 0.3.2*

**import** random

**import** json

In [ ]:

*# importing our intent file used for training the model.*

**with** open("intents.json") **as** json\_data:

intents **=** json**.**load(json\_data) *# Loading data from intents.json file to var intents*

In [ ]:

*# Empty lists for appending the data after processing NLP*

words**=**[]

documents **=** []

classes **=** []

*# This list will be used for ignoring all unwanted punctuation marks.*

ignore **=** ["?"]

*# Starting a loop through each intent in intents["patterns"]*

**for** intent **in** intents["intents"]:

**for** pattern **in** intent["patterns"]:

*# tokenizing each and every word in the sentence by using word tokenizer and storing in w*

w **=** nltk**.**word\_tokenize(pattern)

*#print(w)*

*# Adding tokenized words to words empty list that we created*

words**.**extend(w)

*#print(words)*

*# Adding words to documents with tag given in intents file*

documents**.**append((w, intent["tag"]))

*#print(documents)*

*# Adding only tag to our classes list*

**if** intent["tag"] **not** **in** classes:

classes**.**append(intent["tag"]) *#If tag is not present in classes[] then it will append into it.*

*#print(classes)*

In [ ]:

*#Performing Stemming by using stemmer.stem() nd lower each word*

*#Running loop in words[] and ignoring punctuation marks present in ignore[]*

words **=** [stemmer**.**stem(w**.**lower()) **for** w **in** words **if** w **not** **in** ignore]

words **=** sorted(list(set(words))) *#Removing Duplicates in words[]*

*#Removing Duplicate Classes*

classes **=** sorted(list(set(classes)))

*#Printing length of lists we formed*

print(len(documents),"Documents \n")

print(len(classes),"Classes \n")

print(len(words), "Stemmed Words ")

training **=** []

output **=** []

*#Creating empty array for output*

output\_empty **=** [0] **\*** len(classes)

*#Creating Training set and bag of words for each sentence*

**for** doc **in** documents:

bag **=** [] *#Initialising empty bag of words*

pattern\_words **=** doc[0] *#Storing list of tokenized words for the documents[] tp pattern\_words*

*#print(pattern\_words)*

*#Again Stemming each word from pattern\_words*

pattern\_words **=** [stemmer**.**stem(word**.**lower()) **for** word **in** pattern\_words]

*#print(pattern\_words)*

*#Creating bag of words array*

**for** w **in** words:

bag**.**append(1) **if** w **in** pattern\_words **else** bag**.**append(0)

*#It will give output 1 for curent tag and 0 for all other tags*

output\_row **=** list(output\_empty)

output\_row[classes**.**index(doc[1])] **=**1

training**.**append([bag, output\_row])

In [ ]:

random**.**shuffle(training) *#Suffling the data or features*

training **=** np**.**array(training) *#Converting training data into numpy array*

*#Creating Training Lists*

train\_x **=** list(training[:,0])

train\_y **=** list(training[:,1])

In [ ]:

tf**.**reset\_default\_graph() *#Reset Underlying Graph data*

*#Building our own Neural Network*

net **=** tflearn**.**input\_data(shape**=**[**None**, len(train\_x[0])])

net **=** tflearn**.**fully\_connected(net, 10)

net **=** tflearn**.**fully\_connected(net, 10)

net **=** tflearn**.**fully\_connected(net, len(train\_y[0]), activation**=**"softmax")

net **=** tflearn**.**regression(net)

*#Defining Model and setting up tensorboard*

model **=** tflearn**.**DNN(net, tensorboard\_dir**=**"tflearn\_logs")

*#Now we have setup model, now we need to train that model by fitting data into it by model.fit()*

*#n\_epoch is the number of times that model will se our data during training*

model**.**fit(train\_x, train\_y, n\_epoch**=**1000, batch\_size**=**8, show\_metric**=True**)

model**.**save("model.tflearn") *#Saving the model*

In [ ]:

*#Importing pickle module*

**import** pickle

*#Dumping training data by using dump() and writing it into training\_data in binary mode*

pickle**.**dump({"words":words, "classes":classes, "train\_x":train\_x, "train\_y":train\_y}, open("training\_data", "wb"))

data **=** pickle**.**load(open("training\_data","rb"))

words **=** data['words']

classes **=** data['classes']

train\_x **=** data['train\_x']

train\_y **=** data['train\_y']

In [ ]:

**with** open("intents.json") **as** json\_data:

intents **=** json**.**load(json\_data) *#Loading our json\_data*

In [ ]:

*# Loading the saved model*

model**.**load("./model.tflearn") *#Loading training model which we saved*

In [ ]:

*#Cleaning User Input*

**def** clean\_up\_sentence(sentence):

*# Tokenizing the pattern*

sentence\_words **=** nltk**.**word\_tokenize(sentence) *#Again tokenizing the sentence*

*#Stemming each word from the user's input*

sentence\_words**=** [stemmer**.**stem(word**.**lower()) **for** word **in** sentence\_words]

**return** sentence\_words

*#Returning bag of words array: 0 or 1 or each word in the bag that exists in as we have declared in above lines*

**def** bow(sentence, words, show\_details**=False**):

*#Tokenizing the user input*

sentence\_words **=** clean\_up\_sentence(sentence)

*#Generating bag of words from the sentence that user entered*

bag **=** [0]**\***len(words)

**for** s **in** sentence\_words:

**for** i,w **in** enumerate(words):

**if** w **==** s:

bag[i] **=** 1

**if** show\_details:

print("Found in bag: %s"**%** w)

**return**(np**.**array(bag))

context **=** {} *#Create a dictionary to hold user's context*

ERROR\_THRESHOLD **=** 0.25

**def** classify(sentence):

*#Generating probabilities from the model*

results **=** model**.**predict([bow(sentence, words)])[0]

*#Filter out predictions below a threshold*

results **=** [[i,r] **for** i,r **in** enumerate(results) **if** r**>**ERROR\_THRESHOLD]

*#Sorting by strength of probability*

results**.**sort(key**=lambda** x: x[1], reverse**=True**)

return\_list **=** []

**for** r **in** results:

return\_list**.**append((classes[r[0]], r[1]))

*# return tuple of intent and probability*

**return** return\_list

**def** response(sentence, userID**=**'123', show\_details**=False**):

results **=** classify(sentence)

*#If we have a classification then find the matching intent tag*

**if** results:

*#Loop as long as there are matches to process*

**while** results:

**for** i **in** intents['intents']:

*#Find a tag matching the first result*

**if** i['tag'] **==** results[0][0]:

*#Set context for this intent if necessary*

**if** 'context\_set' **in** i:

**if** show\_details: print ('context:', i['context\_set'])

context[userID] **=** i['context\_set']

*# check if this intent is contextual and applies to this user's conversation*

**if** **not** 'context\_filter' **in** i **or** \

(userID **in** context **and** 'context\_filter' **in** i **and** i['context\_filter'] **==** context[userID]):

**if** show\_details: print ('tag:', i['tag'])

*#A random response from the intent*

**return** print(random**.**choice(i['responses']))

results**.**pop(0)